XBench Benchmark and Performance Testing of XML DBMSs

Benjamin Bin Yao M. Tamer Özsu
University of Waterloo
School of Computer Science
{bbyao, tozsu}@uwaterloo.ca

Nitin Khandelwal*
University of Pennsylvania
Department of Computer & Information Science
khandeln@seas.upenn.edu

Abstract

XML support is being added to existing database management systems (DBMSs) and native XML systems are being developed both in industry and in academia. The individual performance characteristics of these approaches as well as the relative performance of various systems is an ongoing concern. In this paper we discuss the XBench XML benchmark and report on the relative performance of various DBMSs. XBench is a family of XML benchmarks which recognizes that the XML data that DBMSs manage are quite varied and no one database schema and workload can properly capture this variety. Thus, the members of this benchmark family have been defined for capturing diverse application domains.

1. Introduction

XML (eXtensible Markup Language) [3] is beginning to be extensively used in various application domains and significant amounts of XML documents are being generated. This has raised the demand for their efficient management. There has been considerable research on efficiently storing, manipulating, and retrieving XML documents. A number of approaches have been proposed, including using flat file systems (e.g., Kweelt [15]), extending mature DBMS technologies such as relational DBMSs (e.g., IBM DB2, Oracle, and Microsoft SQL Server) or object-oriented DBMS (e.g., Ozone [10]), and building native XML repositories (e.g., Tamino [17], Natix [9], X-Hive[1], and Xyleme[2]). The individual performance characteristics of these approaches as well as the relative performance of various systems is an ongoing concern. In this paper we describe the XBench family of benchmarks for testing XML DBMSs and provide a subset of the performance results that we have obtained by exercising this benchmark on a set of commercial DBMSs (DB2, SQL Server, and X-Hive).

There are several major domain-specific database benchmarks from the Transaction Processing Performance Council (TPC) – in particular TPC-W for Web e-commerce – but these benchmarks do not directly address the requirements of XML databases (e.g., nested document structures and path expression queries). There are a number of XML benchmark proposals and these come in two groups: micro benchmarks and application benchmarks. Micro benchmarks are designed to test individual system components to isolate problems, measure, and, thus, improve a particular component of an XML system. The Michigan Benchmark [14] belongs in this category.

Application benchmarks, on the other hand, measure the overall performance of a DBMS. XBench is an application benchmark. The others in this category are XMach-1 [2], XMark [16], and XOO7 [4].

XMach-1 is a multi-user benchmark that is based on a Web application and considers text documents and catalog data. It defines a small number of XML queries that cover multiple functions and update operations for which system performance is determined. It provides support for DTD and does not consider XML Schema for optimization.

XMark is a single-user benchmark. The database model is based on an Internet auction site, and, therefore, its database contains one big XML document with text and non-text data. Compared to XMach-1, it provides a more comprehensive set of queries, but it has no support for XML Schema.

XOO7 was derived from OO7 [5], which was designed to test the efficiency of object-oriented DBMS. The database model of XOO7 is mapped from that of OO7. Besides mapping the original queries of OO7 into XML, XOO7 adds some XML specific queries. It supports DTDs but not XML Schema.

In summary, all of these benchmarks, except XBench consider only one type of application running against one database schema. Their workloads cover different functionalities, but leave out a number of XQuery [7] features. XBench covers all of XQuery functionality as captured by XML Query Use Cases [6].

*Work done while the author was visiting the University of Waterloo.
[1]http://www.x-hive.com
[2]http://www.xyleme.com

The range of XML applications and the XML data that they manage are quite varied and no one database schema and workload can properly capture this variety. We, therefore, propose a *family* of XML benchmarks, collectively called XBench. XBench is a single machine benchmark that covers a wide range of database designs that are defined according to a classification of applications. It tests for scalability (small (10 MB), normal (100 MB), large (1 GB) and huge (10 GB) databases) and for full XQuery functionality as captured in XML Query Use Cases. A functional comparison of key features on these four XML benchmarks as well as a comparison of their workloads is given in [20].

We describe the XBench benchmark in Section 2. Section 3 discusses the partial results of performance testing that we have performed on three systems using XBench. Finally, we conclude in Section 4 and indicate the current directions that we are pursuing.

2. XBench Benchmark

2.1. Database Design

We conducted detailed statistical analysis of a number of XML data sets (some converted from SGML) [20]. This analysis demonstrated significant differences among the data sets, leading to a classification of XML documents and applications.

Database applications are characterized along two dimensions: application characteristics, and data characteristics. Application characteristics indicate whether they are data-centric or text-centric. *Data-centric* (DC) applications deal with data that may not originally be in XML. Examples include e-commerce catalog data or transactional data that are captured as XML. *Text-centric* (TC) applications manage actual text data natively encoded as XML documents. Examples include dictionaries, book collections in a digital library, or news article archives.

In terms of data characteristics, two classes are identified: single document[3] (SD) and multiple document (MD). The *single document* case covers databases, such as an e-commerce catalog, that consist of a single document with complex structures (deep nested elements), and dictionaries, while the *multiple document* case covers those databases that contain a set of XML documents, such as an archive of news documents or transactional data. Thus, a database generator is needed that can handle four cases: DC/SD, DC/MD, TC/SD, and TC/MD (Table 1).

Many of the real XML documents in DC/SD and DC/MD classes are currently relational data that have been transformed into XML. This may change in the future.

For the purposes of these benchmarks, synthetic data are generated with domain specific features. Real XML documents that best represent their particular classes were analyzed, where available, considering at least two types of documents in each class.

[3] "Document", in this context, refers to an XML document.

	SD	MD
TC	Online dictionaries	News corpus, Digital libraries
DC	E-commerce catalogs	Transactional data

Table 1. Classification & Sample Applications

2.1.1 Text-Centric Document Databases

There are sufficient amounts of real text-centric (TC) XML documents. For the single document class, two dictionaries were analyzed: GNU version of the Collaborative International Dictionary of English (GCIDE)[4] and the Oxford English Dictionary (OED)[5]. For multiple document class, a collection of XML documents that make up the Reuters news corpus was used along with a part of Springer-Verlag digital library that was provided to us. The OED dictionary and the Springer data are in SGML and were converted to XML prior to analysis.

Sources	No. files	File size	Data size (MB)
GCIDE	1	56 MB	56
OED	1	548 MB	548
Reuters	807,000	[1, 59] KB	2,484
Springer	196,000	[1, 613] KB	1,343

Table 2. Analyzed TC Class Data

Statistical data are collected for these documents in terms of a set of parameters: collection of element types, database schema structure (parent/child relationships among element types), probability distribution of instance occurrences of immediate child elements to a parent element, probability distribution of element values to types, collection of attribute names, probability distribution of attribute values to names, and probability distribution of attributes to each element. For each distribution parameter, the minimum and maximum values of that distribution are defined in order to generate finite documents. Based on the statistics, frequency distributions are computed and standard probability distributions are fit to the data.

Since two XML documents are analyzed in each of the TC classes, it is necessary to generalize the common characteristics of that class. We define an XML document structure for each class that is derived from the original XML documents. The process of generalization also involves combining statistical data of two or more semantically same

[4] http://www.ibiblio.org/webster

[5] http://www.oed.com

element types (from same or different XML documents), producing a probability distribution or a combination of several distributions. Due to space considerations, we omit the details and refer the reader to [20].

The common features of XML documents in the TC/SD class are a big text-dominated document with repeated similar entries, deep nesting and possible references between entries. The generated XML document is a single big XML document (*dictionary.xml*) with numerous word entries. The size of the database is controlled by a parameter called *entry_num*. The default value of entry_num is 7333 and file size is about 100 MB.

Figure 1 gives a visual representation of the schema of XML documents in this class[6]. The rectangles refer to element types; solid rectangles mean element types are mandatory while dotted ones mean they may or may not exist in a given document.

The features of XML documents in the TC/MD class are numerous relatively small text-centric XML documents with references between them, looseness of schema and possibly recursive elements. The target XML documents are a set of XML articles called *articleXXX.xml* (XXX represents the number of a particular document) with sizes ranging from several kilobytes to several hundred kilobytes. The size of this database is controlled by *article_num* with a default value of 266, and the default data size is around 100 MB.

Figure 2 illustrates the schema information of XML documents in this class. The figure depicts the irregularity of this class of documents.

2.1.2 Data-Centric Document Database

For data-centric classes, the availability of real XML data for analysis is problematic. Although there are XML specifications of transactional data (e.g., Electronic Catalog XML (eCX)[7], Commerce XML (cXML)[8], XMLPay[9], and XML Common Business Library (xCBL)[10]), they are not yet widely used and no substantial repository of XML documents can be found. The data that is available is too small to extract meaningful statistics. Most of the XML documents in the data-centric classes are currently relational that may be translated into XML for communication. Therefore, the schema of the TPC-W benchmark [18] is used and is mapped to XML. TPC-W considers a Web-based e-commerce system and can represent DC classes of documents.

There are eight basic individual tables (relations) in the TPC-W database: ORDERS (purchase order information), CC_XACTS (information on credit card transaction for each order), ORDER_LINE (information on all detailed items of each order), CUSTOMER (customer information), ITEM (information about TPC-W item, which are books), AUTHOR (author information), ADDRESS (address information for each customer), and COUNTRY.

XML documents belonging to the DC/SD class are similar to TC/SD in terms of structure but with less text content. However, the schemas of these data tend to be more strict in the sense that there is less irregularity in DC/SD than in TC/SD, since most of the XML documents in DC/SD are translated directly from relations.

In order to create a catalog data structure from TPC-W relational data model, table ITEM is picked as base, along with the AUTHOR, ADDRESS and COUNTRY tables. Two more tables are created that do not exist in TPC_W: AUTHOR_2 to include additional author information such as: mailing address, phone and email information, and PUBLISHER to record publisher information consisting of name, fax, phone and email address.

These six tables are joined together and mapped to an XML document called *catalog.xml*. By joining all these tables, more depth is added to *catalog.xml*. In this mapping, we follow an approach that takes into consideration the domain-specific semantics. The mapping of a join of several tables is done by picking one main table (in our case ITEM) and mapping it first to XML using the above described method. All matching tuples in the second table are inserted as sub-elements of the corresponding tuples in the first table based on foreign key references. The process is repeated recursively for other tables. Accordingly, as the number of joined tables increases, the mapped XML document gets deeper[11]. The visual representation of the tree structure of all element types are shown in Figure 3.

Data-centric multiple documents are transactional and are primarily used for data exchange. Thus, the tags are more descriptive and contain less text content. Usually, the structure is more restricted (in terms of irregularity) and flat (less depth) since most of the data originates in relational databases.

Due to the nature of these documents, we use the *flat translation* (FT) [19, 11] approach that maps a relation into an element type. Each tuple in the relation is mapped into an instance of the element type, and all the columns are mapped into sub-elements (if it is attribute-oriented, all columns are mapped into attributes of the element). The resulting XML documents of this method are very flat. FT approach is used to map five of the TPC-W tables (CUSTOMER, ITEM, AUTHOR, ADDRESS

[6]The complete XML Schema and DTD files for all database classes are given in [20].

[7]http://www.ecx-xml.org

[8]http://www.cxml.org

[9]http://www.verisign.com/developer/xml/xmlpay.html

[10]http://www.xcbl.org

[11]There are other mapping techniques that have been proposed such as flat transaction [19, 11], nesting-based translation [11], and constraints-based Translation [12]. Review of the problems in using these for our purposes is given in [20].

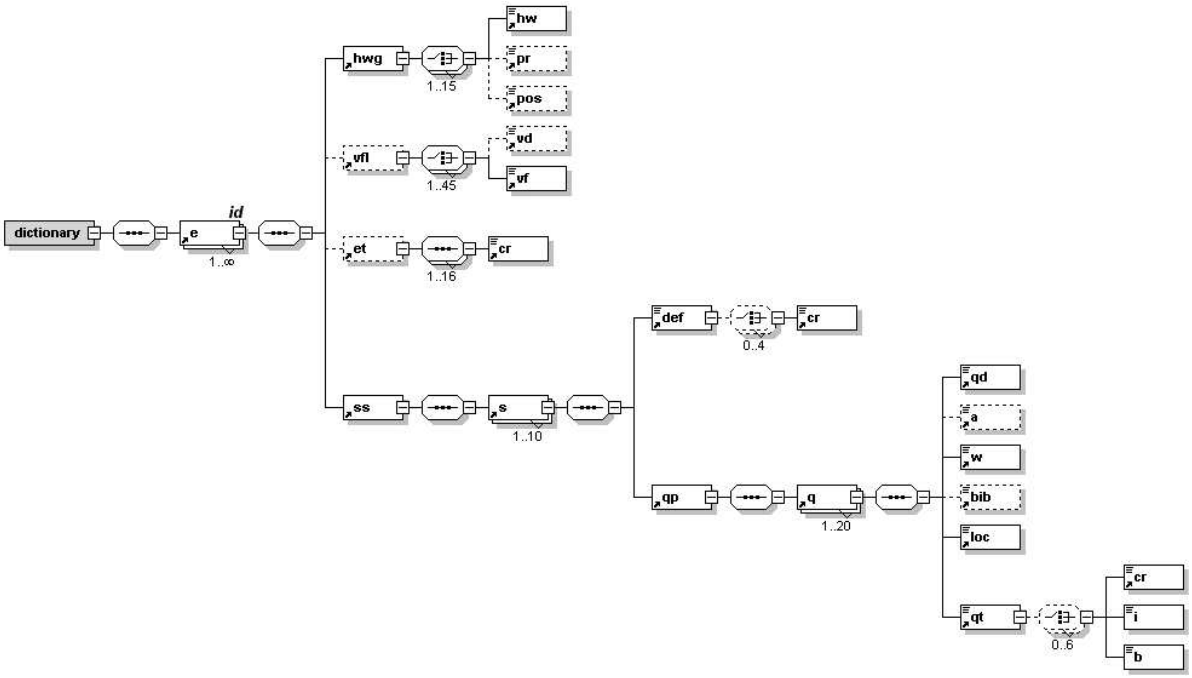

Figure 1. Schema Diagram of TC/SD (Dictionary)

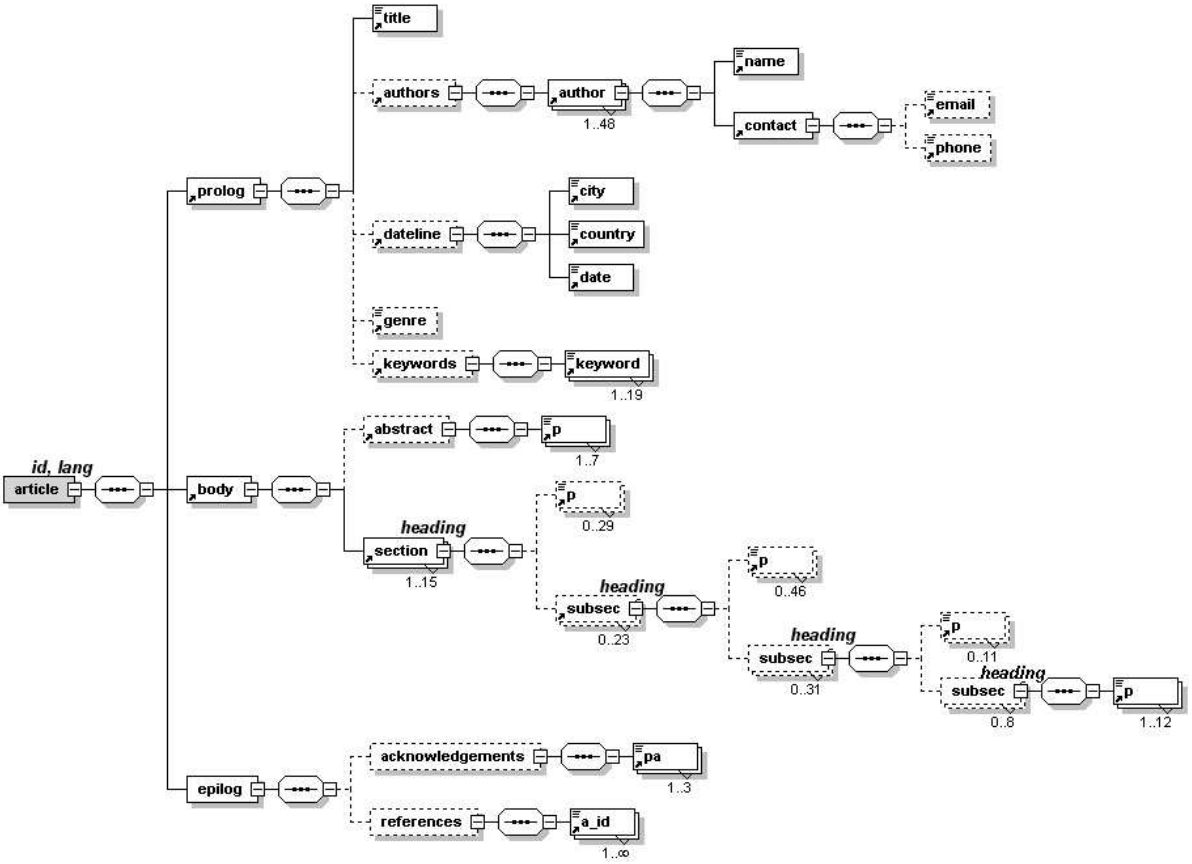

Figure 2. Schema Diagram of TC/MD (ArticleXXX)

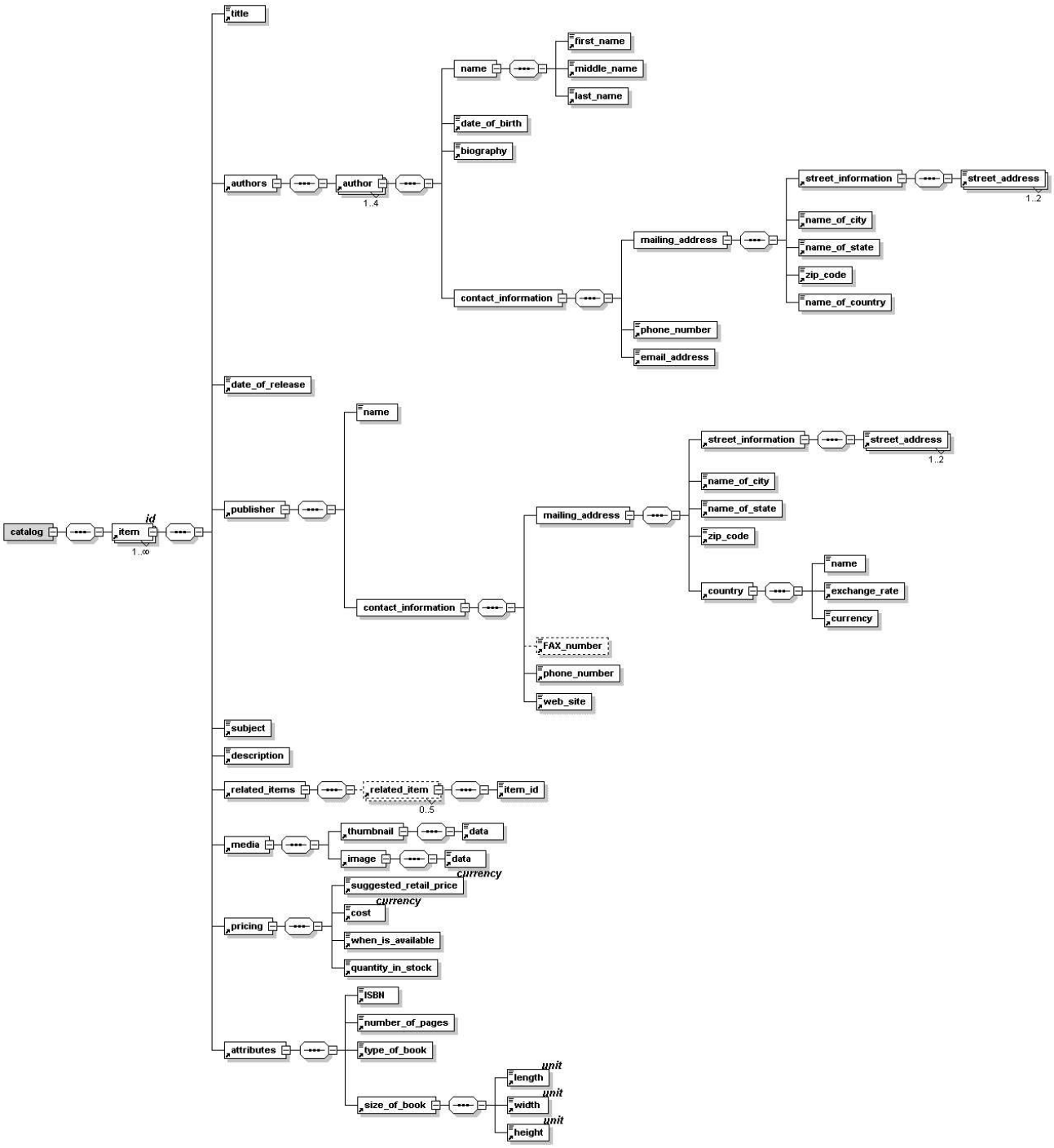

Figure 3. Schema Diagram of DC/SD (Catalog)

and COUNTRY) to separate XML documents, called *Customer, Item, Author, Address,* and *Country*, respectively. Tables ORDERS, ORDER_LINE and CC_XACTS (tables ORDERS and ORDER_LINE have one-to-many relationship and tables ORDERS and CC_XAVTS have one-to-one relationship), are joined together into one big table and mapped into multiple XML documents called *orderXXX.xml*. Each of these documents contains exactly one order information that comes from the three tables. The structure of this XML document (*orderXXX.xml*) is given in Figure 4. The schemas of the other XML documents in this calss are omitted due to space restrictions and can be found in [20].

2.1.3 Database Generation

For actual data generation, we use ToXgene [1], which is a template-based tool facilitating the generation of synthetic XML documents. A ToXgene template is created for each of the four classes discussed above.

2.2. Workload Design

In this first version of XBench, we only focus on queries and bulk loading; workloads testing update performance will be included in subsequent versions. XBench workload covers the full XQuery functionality as captured in XML Query Use Cases. The queries are specified abstractly to demonstrate the functionality that they capture; each of these abstract queries are mapped to specific queries according to the application class (see [20]), generating four workload classes. In the remainder, we discuss the functionality covered by the queries in the workload and give examples from the various document classes[12].

Altogether, there are 20 query types, but each workload class does not contain all of them. XQuery specification of each query is given in [20]. Readers should consult the database schemas given in the previous section to better understand each of the queries.

- **Exact match.** These queries require string exact match with specified and possibly long path expressions, depending on the levels of predicates being queried in XML documents. Consequently, they can be *shallow* queries, that match only at the top level of XML document trees (example query Q1), or *deep* queries that match the nested structure of XML document tree (query Q2).
  Q1 (DC/SD): *Return the item that has matching item id attribute value X.*
  Q2 (TC/MD): *Find the title of the article authored by Y.*
- **Function application.** These queries challenge the system with aggregate functions such as *count*, *avg*, *max*, *min* and *sum*.

[12]For each example query, its workload class is specified.

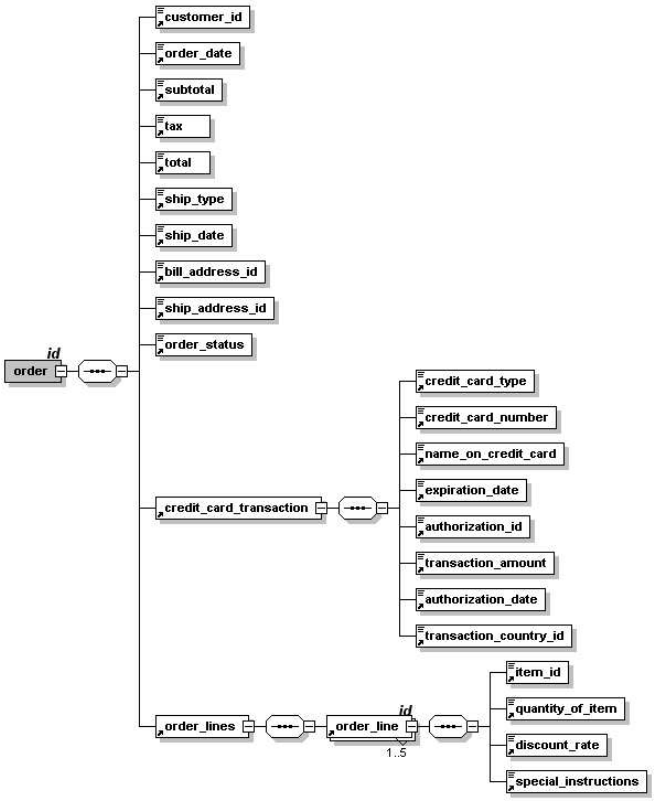

Figure 4. Schema Diagram of DC/MD (OrderXXX)

Q3 (TC/SD): *Group entries by quotation location and calculate the total number of entries in each group.*

- **Ordered access.** These queries test the performance of the system when it preserves the document order during retrieval. This could be *relative order* (Q4) based on the current matching position, or *absolute order* (Q5), which is the order in the document.
  Q4 (TC/MD) *Find the heading of the section following the section entitled "Introduction" in articles written by Y.*
  Q5 (DC/MD) *Return the first order line item of a certain order with id attribute value X.*
- **Quantification.** The two cases are the existentially (Q6) and universally (Q7) quantified queries.
  Q6 (TC/MD) *Find titles of articles where two keywords "K1" and "K2" are mentioned in the same paragraph.*
  Q7 (DC/SD) *Return item information where all its authors are from country Z.*
- **Path expressions.** Two types of queries are defined involving path expressions: (a) queries that contain path expressions where one element name in the path is unknown (Q8), and (b) queries where multiple consecutive element names are unknown (Q9).

Q8 (TC/SD) *Return quotation text of word "word_1".*
Q9 (DC/MD) *Return the order status of an order with id attribute value X.*

- **Sorting.** Even though the generic data type of element content in XML documents is string, users may cast the string type to other types. Therefore, these queries test the system in sorting both the string types (Q10) and non-string types (Q11).
  Q10 (DC/MD) *List the orders (order id, order date and ship type), sorted by ship type, ordered within a certain time period.*
  Q11 (TC/SD) *List the quotation authors and quotation dates, sorted by date, for word "word_2".*
- **Document construction.** Structure is important in many XML documents. However, some systems experience difficulties in even preserving the document's original structure. This class of queries test the performance of the system in preserving the structure (Q12) and in transforming the structure (Q13).
  Q12 (DC/SD) *Get the mailing address of the first author of item with id attribute value X.*
  Q13 (TC/MD) *Extract information from the article that has a matching id attribute value Y, including title, the name of the first author, date, and abstract.*
- **Irregular data.** Irregularity of schema is a fact of life in XML databases. These quries test missing elements (Q14) and empty (null) values (Q15).
  Q14 (DC/SD) *Return the names of publishers who publish books in a given time period but do not have a fax number.*
  Q15 (TC/MD) *List author names whose contact elements are empty in articles published within a certain time period.*
- **Retrieval of individual documents.** This query tests an essential function of an XML DBMS to retrieve individual XML documents efficiently while preserving the contents of those documents.
  Q16 (DC/MD) *Retrieve one whole order document with an id attribute value X.*
- **Text search.** These queries test the information retrieval capabilities of systems. Two cases are tested: *uni-gram search* (Q17) where the query contains one particular word, and *bi-gram* and *n-gram search* (Q18) where multiple words are involved.
  Q17 (TC/SD) *Return the headwords of the entries that contain the word "word_x".*
  Q18 (TC/MD) *List the titles and abstracts of articles that contain the phrase "...".*
- **References and joins.** Data-centric documents usually have references to identify the relationship between related data, even among different XML documents. Sometimes users want to combine separate information together using join by values. These queries test this feature.
  Q19 (DC/MD) *For a particular order with id attribute value X, get its customer name and phone, and its order status.*
- **Datatype casting.** The element values in XML documents are String type, but sometimes they need to be cast into other data types.
  Q20 (DC/SD) *Retrieve the item title whose size is larger than a certain number.*

3. Performance Experiments

We have used XBench to test a number of commercial DBMSs. In this section we summarize some of the findings over three systems: IBM DB2 UDB version 8.1 (hereafter referred to as DB2), Microsoft SQL Server 2000 SP3 (referred to as SQL Server) and X-Hive/DB 4.1.1 (referred to as X-Hive). The full set of results are given in [13].

3.1. Experimental Setup

All experiments are conducted on a 2Ghz Pentium machine with 1GB main memory and 60GB disk running Windows XP. We report the performance of systems for the small, normal and large database sizes for each database domain (TC/SD, TC/MD, DC/SD, and DC/MD). In our experiments, we create separate database instances for all the scenarios. For example, we create three database instances for TC/SD, called TCSDS (for small), TCSDN (for normal) and TCSDL (for large). The workload for the experiments reported in this paper consists of queries Q5, Q8, Q12, Q14, and Q17. For DB2 and SQL Server, the queries are converted into SQL since they do not support an XQuery interface.

Classes	Indexes
TC/SD	hw
TC/MD	article/@id
DC/SD	item/@id, date_of_release
DC/MD	order/@id

Table 3. Indexes for Each Class

For fairness, we only create value indexes on the elements/attributes that are most frequently used by the queries in each document class, and can be implemented for all systems. We also take vendor recommendations into consideration in the case of X-Hive. Table 3 shows the indexes that are relevant to the queries reported in this paper. All arbitrary indexes are created separately after bulk loading, except that for relational DBMSs, the indexes are created automatically for primary/foreign keys of some tables.

The execution time of a query is the cold run time (from the time when a user submits a request, to the time when a user gets the full result) to prevent caching effects. We measure two times for each query: with no indexes (i.e., sequential scan) to form a baseline, and with indexes. We only report, in the following tables, times with indexes.

3.1.1 DB2

We use XML Extender, which is a component included in DB2 to support storage and retrieval of XML documents, to map XML schemas to DB2. There are two options: XML column (referred as Xcolumn) and XML collection (referred as Xcollection). We use both options in our experiments.

In the case of Xcolumn, an entire XML document is kept as a CLOB in an Xcolumn of a table. A couple of side tables are created for searchable elements/attributes. A column called `dxx_seqno` is also added to each side table, in the cases where elements have multiple occurrences, in order to keep the ordering information. Data Access Definition (DAD), which is an annotated XML file, is used to define which elements/attributes are searchable. Xcolumn is perfect for application domains where there are multiple small XML documents. It cannot deal with the situation where there is a single large XML document without the help of Text Extender (a different IBM product), since the upper limit of an XML CLOB is 2GB. Therefore, in our experiments we use Xcolumn only for DC/MD and TC/MD classes.

Xcollection shreds XML documents into one or multiple tables according to DADs explicitly stated by users. The mapping is very similar to the mapping described in Section 2.1.2, except that it is backwards. The DAD file, in this case, is used to describe the mappings from elements/attributes to columns in tables. Xcollection shreds a large XML document into multiple tables, so that the system can deal with queries efficiently. We use Xcollection for all classes.

XML Extender can check whether an XML document is well-formed during the loading phase, and does not make use of DTD or XML Schema meta-data to store and index XML documents.

3.1.2 SQL Server

There are two approaches to loading XML data to SQL Server. The first alternative uses stored procedures to load an XML document into main memory and pre-process it. The document is then shredded and stored in the database. This approach requires the document to be loaded into memory for pre-processing, thus restricting the file size. Many of the documents that are in XBench are too big to be loaded in this manner.

The second approach, which is the one we use in these experiments, is to use the XML bulk loading tool of SQLXML (Version 3.0 SP1) that allows storage of XML documents by shredding them into one or multiple tables. An annotated XML Schema Definition (XSD) can be defined explicitly to describe mapping or the mapping can be generated implicitly in the absence of schema information. A VBScript is used to load XML documents into SQL Server databases.

3.1.3 Problems with Relational Mappings

There are a number of problems in mapping the XML schemas to relational systems. We discuss these problems below and indicate, in each case, which system has the problem.

1. It is difficult to distinguish between elements and attributes (DB2 Xcollection and SQL Server).
2. It is difficult to deal with and maintain the ordering of child elements (DB2 Xcollection and SQL Server)[13].
3. It is not possible to properly map mixed content elements. We have to ignore these elements with mixed contents, such as the element *qt* in *dictionary.xml* (SQL Server).
4. It is difficult to map chain relationships where the elements on the chain do not have unique values (DB2 Xcollection and SQL Server[14]). For example, consider the following XML document fragment:

   ```
   <article id="1">
      <sec>
         <p>...</p>
         <p>...</p>
      </sec>
      <sec>
         <p>...</p>
         <p>...</p>
      </sec>
   </article>
   ```

   Since element `sec` does not have any unique value, it is difficult to keep track of its ordering information, and even worse, it is impossible to tell which `sec` a particular `p` belongs to after the document is shredded. We solve this by adding a unique `id` attribute to those elements that have this problem.
5. Another issue about DB2 Xcollection is that the maximum number of rows in a table is restricted to 1024 for a decomposed XML document. This number is fairly small when it comes to big XML documents. Even if

[13] It is possible to solve the first two problems by adding extra columns, tables and triggers, but it is not very practical and imposes performance penalties. Therefore, we do not fix these issues in our experiments. Furthermore DB2 Xcolumn handles order via the `dxx_seqno` column that is added to each side table as discussed in Section 3.1.1.

[14] In the case of SQL Server the issues related to ordering and chain relationships do not arise if stored procedure approach is used.

	DC/SD			DC/MD			TC/SD			TC/MD		
	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large
Xcolumn	-	-	-	30	417	11532	-	-	-	12	85	662
Xcollection	34	-	-	87	1126	31860	46	-	-	40	124	762
SQL Server	43	120	770	119	1438	39496	55	153	960	52	148	894
X-Hive	9	59	517	25	304	8568	12	72	647	7	57	512

Table 4. Bulk Loading Time (in Seconds)

the document itself is not very big, if there is a leaf element with multiple instances, the limit will be reached easily. One possible solution is to split the original XML document into smaller documents before loading them into the database. However, during our experiments, we found that a 10 MB XML document almost exceeds this limit. For a 1GB document, we have to split them into approximately 100 files, which is not practical. Therefore, we only have experiments on the databases of size 10 MB for the classes of TC/SD and DC/SD.

6. DB2 Xcolumn solves these issues naturally but suffers from large size XML documents. We only have experiments on the classes of TC/MD and DC/MD.

The end result of these issues is that some of the queries that deal with the issues raised above may not generate correct results, even though we report their performance.

3.1.4 X-Hive

X-Hive is a Java-based native XML DBMS supporting XQuery. Since X-Hive is a native XML DBMS, there is no need for any database mapping. For bulk loading XML documents, it is possible to use XHAdmin utility. In our experiments, the utility's GUI interface became sluggish after loading the 100M database. We, therefore, used the Java API that X-Hive provides to load the databases.

3.2. Results

In this section we describe the results of the experiments. In order to ensure that all implementations are equivalent in the design of the databases and that the benchmark is applied as it was intended, the implementation of the databases, and the query translations from XQuery to their own languages (when required) were done by us.

3.2.1 Experiment 1: Complexity of Mapping and Bulk Loading Time

This experiment concerns the efficiency of loading XML documents into each DBMS. For relational DBMSs, there is the additional issue of mapping XML into relational tables and the effect that this would have on the bulk loading time. We turn off the option of conforming/validating XML documents (if a DBMS provides such functionality) during bulk loading to reduce time costs. Our results that are grouped by database type and then by database size are reported in Table 4.

As expected, the bulk loading times of two relational DBMSs are much longer than that of X-Hive because they need extra time to shred documents and create indexes for primary/foreign keys. Data-centric documents generally are loaded faster than text-centric documents due to the relative simplicity of data-centric document structures. For a given database size, the bigger the individual files, the slower is the loading time. However, the number of documents becomes very critical once the number becomes very high. All DBMSs load data much slower in DC/MD than others for the same database size, due to the fact that DC/MD has many more XML documents than the others, requiring extra I/O operations. When the database size increases 10 times, the loading time does not necessarily increase linearly except for DC/MD, in which case, the number of files (e.g., over 200k for 1GB database size) dominates the loading time.

3.2.2 Experiment 2: Document Structure and Its Physical Storage

XML enabled DBMSs store XML documents using one of the following three approaches: storing an XML document as a whole, shredding an XML document into fragments, and combining the above two approaches. The first approach works efficiently on ordered access and retrieving big fragments or whole documents, but performs poorly on text search. The second approach works well on text search but is problematic on ordered access and document reconstruction. The last approach suffers high space cost and redundancy. This experiment is designed to investigate this issue and includes queries Q5, Q12 and Q17.

The ordering of elements is important in XML documents. Although XML documents are stored in relational DBMSs, the orders should still be preserved. Query Q5 is designed to return data based on its order in a document. Table 5 shows the execution time in milliseconds of query Q5 in each scenario.

One of the big problems experienced by relational DBMSs in storing XML documents is their poor performance on document reconstruction. Depending on the approach used in storing XML documents in DBMSs, some

	DC/SD			DC/MD			TC/SD			TC/MD		
	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large
Xcolumn	-	-	-	90	1598	9567	-	-	-	10	10	15
Xcollection	10	-	-	10	10	15	85	-	-	20	40	65
SQL Server	15	20	25	10	10	20	90	594	3754	20	45	70
X-Hive	10	10	20	335	7460	213347	20	901	30886	30	60	80

Table 5. Query Q5 Execution Time (in Milliseconds)

	DC/SD			DC/MD			TC/SD			TC/MD		
	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large
Xcolumn	-	-	-	30	1487	7631	-	-	-	15	20	25
Xcollection	20	-	-	10	10	15	85	-	-	70	403	3101
SQL Server	20	25	30	10	10	20	90	587	3792	80	458	3318
X-Hive	30	50	50	105	911	76280	10	201	43294	60	165	195

Table 6. Query Q12 Execution Time (in Milliseconds)

systems cannot even preserve the document's original structure. However, structure is very important to XML documents, especially to text documents. Query Q12 retrieves fragments of original documents with original structures. The execution time in milliseconds of query Q12 in each scenario is described in Table 6.

Text search plays a very important part in XML document systems. The integration of information retrieval (IR) technologies with database querying is an emerging area of study. Regardless of the storage mechanism that a DBMS uses, the system should perform well on uni-gram, bi-gram and n-gram search. Query Q17 returns XML documents that contain a particular word. Table 7 depicts the execution time in milliseconds of query Q17 in each scenario.

Two relational systems take advantage of the indexes on primary/foreign keys to speed up execution of query Q5. DB2/Xcolumn can keep track of ordering information by using `dxx_seqno`. DB2/Xcollection and SQL Server happen to return correct results for this particular query but they do not guarantee correctness since they do not maintain document order nor do they store any ordering information.

X-Hive performs very well in XML document construction (Q12); it produces the correct result in a short time. Other systems' responses are relatively slow. Since XML documents are shredded and stored in tables, numerous joins are required in order to re-construct the document. Even worse, the structure of the resulting document fragment is not necessarily the same as the original. DB2/Xcolumn is an exception because it stores XML documents intact.

For Q5 and Q12, creating necessary indexes on some commonly used elements speeds up the queries, specially for X-Hive, since two relational DBMSs automatically create some indexes on primary/foreign keys when bulk loading, which may (most likely) cover those elements. The indexing does not make a big difference for small databases, but start to take positive effects when the databases get larger.

None of the systems does well on Q17 that involves text search. X-Hive supports the creation of full text indexes on particular elements. We do not use it in our experiments because we cannot build similar full text indexes for the relational systems. Use of full-text index will certainly improve its performance. For relational DBMSs, when elements are shredded into many tables, creating full-text indexes on these elements means creating indexes on all columns in the tables. Furthermore, DB2 and SQL Server have limits for indexing on the size of columns and most likely those columns are too big to create indexes on. IBM has a different product, called Text Extender, which is targeted to this issue, but this is not used in our experiments.

Overall, in this experiment, DB2/Xcollection and SQL Server perform better than X-Hive in data-centric and text-centric/single document domains, especially in large database sizes (beyond 100MB). X-Hive and DB2/Xcolumn do better in text-centric/multiple document domain. DB2/Xcollection does slight better than SQL server.

Based on this experiment, as expected, relational DBMSs have a more optimized storage mechanism, particularly for large tables. Since we map multiple *orderXXX.xml* documents into two tables (*order_tab* and *order_line_tab*) in DC/MD case, relational systems can take advantage of the efficient storage mechanism. On the other hand, X-Hive suffers from accessing huge amounts of XML documents in DC/MD case. For TC/MD domain, there is a limited number of small documents, and that is where X-Hive shows its advantage.

	DC/SD			DC/MD			TC/SD			TC/MD		
	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large
Xcolumn	-	-	-	10	8649	54287	-	-	-	100	856	7859
Xcollection	25	-	-	20	187	1754	90	-	-	95	592	4418
SQL Server	40	304	3194	55	216	1918	95	675	4654	100	634	4593
X-Hive	351	4336	49962	140	8512	249809	711	9023	127974	20	120	1532

Table 7. Query Q17 Execution Time (in Milliseconds)

	DC/SD			DC/MD			TC/SD			TC/MD		
	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large
Xcolumn	-	-	-	20	454	1870	-	-	-	25	187	422
Xcollection	15	-	-	10	10	15	70	-	-	10	10	15
SQL Server	15	20	25	10	10	20	75	436	2537	10	10	20
X-Hive	10	20	20	245	5207	168162	10	120	48459	10	20	50

Table 8. Query Q8 Execution Time (in Milliseconds)

X-Hive also outperforms the other two systems when it comes to reconstruction of document fragments in text-centric domains if original XML documents are not very big (less than 1GB). It should be noted that since the mapping does not maintain the document order or the mixed content elements, the results returned by those two relational DBMSs for Q5 and Q12 are not necessarily accurate.

3.2.3 Experiment 3: Path Expression and Loose Schema

This experiment focuses on the access to XML documents using path expressions and irregularity of XML documents. It involves queries Q8 and Q14.

Even though some relational DBMSs do not directly support path expressions, they should be able to process them after the expressions are translated into SQL [8] or system specific languages. It is common that one or more elements in a path expression are unknown, and query Q8 tests this case. Table 8 reports the execution time of query Q8 for each scenario.

Sometimes the elements in a path expression are not fully known or elements in an XML document are missing. This is due to the fact that XML documents do not have as strict schemas as their relational counterparts. XML schemas are more flexible and may have a number of irregularities such as missing elements and empty values. It is a challenge for relational DBMSs to store such loose schema data. Smart DBMSs should be able to exploit XML schemas associated with XML documents in dealing with irregularity. Query Q14 covers missing elements in XML documents case. The execution time of query Q14 is represented in Table 9.

The results show that, in most cases, relational DBMSs do well on Q8. Recall that in those cases, XQuery are mapped into SQL, and hence, no real path expressions are actually involved in executing the query. The results of these two queries are very similar. X-Hive, on that hand, needs to deal with path expressions and its execution times are close to those of the relational systems.

All relational DBMSs suffer from table scanning for Q14, due to the fact that we do not purposely create any index on that particular missing element, while X-Hive does pretty well in some cases even though there is no index.

X-Hive outperforms DB2 and SQL Server in text-centric categories but performs badly on large size databases of data-centric domains which are the strong points of DB2 and SQL Server.

4. Conclusions

In this paper, we describe XBench, which is a comprehensive XML database benchmark that covers a large number of XML database applications. These applications are characterized by whether they are data-centric or text-centric and whether they consist of a single document or multiple documents. XBench workload covers the functionality of XQuery as captured in the Use Cases.

We also report on the performance of two relational DBMSs (DB2 and SQL Server) and a native XML DBMS (X-Hive) on a subset of the XBench workload and database size. The workload subset that we use represent some of the common queries over XML data.

This first version of XBench has a number of limitations that will be addressed in subsequent releases. The planned extensions are the following: (1) support for distributed environments, (2) update workloads, and (3) more realistic data-centric database design based on real data rather than TPC-W benchmark. We also need to scale up to higher

	DC/SD			DC/MD			TC/SD			TC/MD		
	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large	Small	Normal	Large
Xcolumn	-	-	-	10	143	398	-	-	-	25	477	1950
Xcollection	30	-	-	50	1343	12432	55	-	-	30	165	1685
SQL Server	30	223	2386	193	1520	14318	55	353	2256	40	172	1793
X-Hive	90	2693	40398	210	9764	248067	171	1372	15032	20	20	231

Table 9. Query Q14 Execution Time (in Milliseconds)

database sizes, but this will require re-writing the data generation engine.

Acknowledgements

The Reuters Corpus is provided by Reuters Limited (UK). The Springer Digital Library data is provided by Springer-Verlag GmbH (Germany).

This research was funded in part by a grant from the Natural Sciences and Engineering Research Council (NSERC) of Canada, and by a scholarship to the first author by IBM Canada.

References

[1] D. Barbosa, A. Mendelzon, J. Keenleyside, and K. Lyons. ToXgene: A Template-Based Data Generator for XML. In *Proc. 5th Int. Workshop on the World Wide Web and Databases (WebDB)*, pages 49–54, June 2002.

[2] T. Böhme and E. Rahm. XMach-1: A Benchmark for XML Data Management. In *Proc. German Database Conference (BTW)*, pages 264–273, March 2001.

[3] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition). http://www.w3.org/TR/2000/REC-xml-20001006, October 2000. World Wide Web Consortium (W3C).

[4] S. Bressan, M. Lee, Y. G. Li, Z. Lacroix, and U. Nambiar. The XOO7 XML Management System Benchmark. Technical Report TR21/00, National University of Singapore, CS Department, November 2001.

[5] M. Carey, D. DeWitt, and J. Naughton. The 007 Benchmark. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 12–21, June 1993.

[6] D. Chamberlin, P. Fankhauser, M. Marchiori, and J. Robie. XML Query Use Cases. http://www.w3.org/TR/xmlquery-use-cases. World Wide Web Consortium (W3C).

[7] D. Chamberlin, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery: A Query Language for XML. http://www.w3.org/TR/xquery. World Wide Web Consortium (W3C).

[8] D. DeHaan, D. Toman, M. P. Consens, and M. T. Özsu. A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 623–634, June 2003.

[9] T. Fiebig, S. Helmer, C.-C. Kanne, J. Mildenberger, G. Moerkotte, R. Schiele, and T. Westmann. Anatomy of a Native XML Base Management System. Technical Report TR-02-001, Universität Mannheim, January 2002.

[10] T. Lahiri, S. Abiteboul, and J. Widom. Ozone: Integrating Structured and Semistructured Data. In *Proc. 7th Int. Conf. on Database Programming Languages*, pages 297–323, September 1999.

[11] D. Lee, M. Mani, F. Chiu, and W. Chu. Nesting-Based Relational-to-XML Schema Translation. In *Proc. 4th Int. Workshop on the World Wide Web and Databases (WebDB)*, pages 61–66, May 2001.

[12] D. Lee, M. Mani, F. Chiu, and W. Chu. Effective Schema Conversions between XML and Relational Models. In *Proc. European Conf. on Artificial Intelligence (ECAI), Knowledge Transformation Workshop*, July 2002.

[13] M. T. Özsu and B. B. Yao. Evaluation of DBMSs Using XBench Benchmark. Technical Report CS-TR-2003-24, School of Computer Science, University of Waterloo, Waterloo, Canada, August 2003. Available from http://db.uwaterloo.ca/∼ddbms/projects/xbench.

[14] K. Runapongsa, J. M. Patel, H. V. Jagadish, and S. Al-Khalifa. The Michigan Benchmark. Technical report, University of Michigan, 2002. Available from http://www.eecs.umich.edu/db/mbench/.

[15] A. Sahuguet. KWEELT, the Making-of: Mistakes Made and Lessons Learned. Technical report, Department of Computer and Information Science, University of Pennsylvania, November 2000.

[16] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML Benchmark Project. Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April 2001.

[17] H. Schöning and J. Wäsch. Tamino - An Internet Database System. In C. Zaniolo, P. C. Lockemann, M. H. Scholl, and T. Grust, editors, *Advances in Database Technology - EDBT 2000, Proc. 6th Int. Conf. on Extending Database Technology*, volume 1777 of *Lecture Notes in Computer Science*, pages 383–387. Springer, 2000.

[18] Transaction Processing Performance Council (TPC). TPC Benchmark W (Web Commerce) Specification, Version 1.8. Technical report, February 2002. Available From http://www.tpc.org/tpcw.

[19] V. Turau. Making Legacy Data Accessible for XML Applications. Available from http://www.informatik.fh-wiesbaden.de/∼turau/veroeff.html, 1999.

[20] B. B. Yao, M. T. Özsu, and J. Keenleyside. XBench - A Family of Benchmarks for XML DBMSs. Technical Report CS-TR-2002-39, School of Computer Science, University of Waterloo, Waterloo, Canada, December 2002. Available from http://db.uwaterloo.ca/∼ddbms/projects/xbench.